Adaptive Groupware for Wireless Networks

Tara Whalen and James P. Black
University of Waterloo
Department of Computer Science
Waterloo, Ontario, Canada N2L 3G1
tjwhalen@ccnga.uwaterloo.ca
jpblack@uwaterloo.ca

Abstract

Current groupware systems do not work well over wireless networks, as a wireless connection is of variable and often poor quality. Our research suggests a number of changes that allow groupware applications to make efficient use of a wireless channel. One fundamental change is providing adaptability, which allows applications to take advantage of favourable network conditions and to react to difficult conditions as they occur. Our experiments with a multi-user text editor revealed several strategies for adaptive groupware. Applying these strategies demonstrated a bandwidth reduction in one scenario of over 90%.

Introduction

The development of software that supports collaboration (groupware) has allowed people to work as a group on such projects as the writing of a document, even if they are separated in space or time. With the recent rise in wireless and mobile computing, more collaborators will be communicating over wireless links. Portable computers and wireless networks have allowed individuals more “location independence” in accessing computing resources. It would be helpful to extend this independence to support groups that cannot use a wired network.

Although this idea is promising, current collaborative applications are designed to use the rich capabilities of a wired network. Collaborative work often requires low response times and group awareness features to support distributed interaction, but these requirements place heavy demands on a wireless link. A wireless connection is often poor, suffering from low bandwidth and frequent disconnections. If network traffic becomes heavy, users are subjected to long delays or connection loss. Thus, there is a trade-off between extensive features and short response times. To reduce delay, groupware must be able to scale back its network resource consumption, a change which leads to a less powerful, but still fully operable, version of the application.

Groupware should also be able to adapt to the quality of the wireless connection, which can vary greatly. A mobile user may plug into a wired network, or may move into an area with a better wireless channel. In these cases, the user should benefit from the full features of a groupware application, rather than continuing to use the reduced set. Thus, collaborative applications should be *adaptive*, modifying their behaviour as the quality of the communication environment changes.

This research explores how current real-time groupware applications can be changed to provide better support for collaboration over wireless networks. A multi-user text editor was studied to determine the bandwidth requirements of various features. We then proposed changes that would allow it to work more effectively in a poor wireless environment. These changes primarily involved the group interface, which could be restricted as necessary to allow for faster responses when the channel quality was very poor. An adaptive component was also considered, to allow the interface to return to full capacity when conditions permitted. Modifications to existing groupware can be very simple, but can yield considerable improvements in performance. Preliminary test results and extrapolations suggest that these changes would allow large reductions in network traffic without restricting the interface unnecessarily. These changes would allow the editor to be used effectively under various circumstances, with minimized delays and disruptions. Many of the general design principles developed in this test case can be applied to other types of groupware for a wireless or mobile environment.

Wireless collaboration

There are a number of problems specific to groupware that occur in a wireless environment. Collaboration requires tools for group awareness as well as the sharing of data, features that can consume large amounts of bandwidth. Over a wireless channel with restricted bandwidth, users experience long response times that can lead to a disjointed collaboration session. Wireless networks also have high error rates that cause data to be corrupted or lost; this also lengthens response times because packets must be retransmitted.

Disconnections can lead to lost work and degraded session coordination. A peripheral consideration is that if a groupware application consumes a lot of bandwidth, other networked applications may experience longer delays.

The principal solutions to these problems are two: reduce delays and handle disconnections gracefully. This research looks at the problem from the application layer: what changes can be made to a program to allow it to make the most efficient use of a variable and often poor wireless channel?

Because the application cannot change such factors as bandwidth and disconnection frequency, it must cope with these problems as best it can. Thus, the only way to reduce delays at this level is to reduce network traffic. To make the most effective use of the channel, it is best to send the most useful data, and cut down on redundant, irrelevant, or unimportant data. This ensures that useful data gets through with minimal delay.

Unfortunately, disconnections cannot be reduced by changing the application. Instead, the application should handle disconnections gracefully. Groupware must ensure that sessions move as smoothly as possible despite the periodic disconnection of collaborators.

This paper suggests five areas where groupware applications can be changed to reduce bandwidth and accommodate network disconnections. These are document overviews, incremental updates, concurrency control, document backups, and restricting features. After examining the bandwidth usage of an existing groupware application, we will describe these five areas and show how they can be applied to this application.

Case study: Calliope

In our investigation of adaptive groupware, a real application was analysed to characterize its bandwidth usage and thereby reveal possible improvements for wireless use. The application that was investigated was Calliope, a GroupKit-based multi-user text editor developed by Alex Mitchell at the University of Toronto [8]. Collaborative sessions are replicated, with every user having their own copy of Calliope. A consistent document is maintained by sending updates to every user's copy when any change is made. Calliope also has a rich group interface, with several features that coordinate actions by providing excellent awareness of group actions on the shared workspace.

Application Description

Calliope has six major group interface features that facilitate collaboration: access control, external annotations, public annotations, private text, unstructured text space, and awareness tools. Access control is maintained using multi-level locking: locks can be requested for a word, line, paragraph, or other text selection, each of which grants a user exclusive access to the locked region. External annotations allow users to embed a Universal Resource Locator (URL) in the text; clicking on the external annotation label will spawn a web browser and bring the user to that URL. Public annotations are used to leave a message for other group members or to suggest changes or additions; users click on an icon to have the annotation text presented to them. Private text spaces allow users to create and modify their writing before their work is presented publicly to the group.

Calliope also provides an unstructured text space away from the main document, to support experimentation while writing. It allows for cutting and pasting text from the main document into a scratchpad, for writing and moving text, and for freehand drawing with the mouse. Finally, to support awareness of other users' actions, Calliope provides several mechanisms, including a telepointer and a group scrollbar. The telepointer is a gesturing tool which displays all the users' mouse movements on the shared workspace, while the group scrollbar tracks the location of other users in a large document.

Resource Consumption

In order to see how Calliope might perform over a limited channel, tests were conducted on eight features: telepointer, public and external annotations, keystroke updates, private text space, group scrollbar, locks, and the scratchpad. The tests were designed to determine the amount of traffic being generated by these features.

The testing environment consisted of a two-user Calliope session conducted over a 10 Mbps Ethernet connection. The traffic generated during the session was recorded using `iptrace`, a Unix utility that provides detailed information about network packets.

The results of the traffic traces are based on two users only, but an examination of Calliope's underlying multicast mechanism reveals that for a particular wireless link, the communication traffic scales linearly with the number of users. Bi-directional tests were performed to look for asymmetries in the communication, but none were revealed. Thus, the results below present a single value for each feature test which is the average of a pair of bi-directional tests.

Calliope transmits the commands that are to be executed on remote machines over TCP sockets. This data stream may be broken up into several TCP packets, and each packet requires a TCP acknowledgment. Because an asynchronous communication model is used, there are no application-level acknowledgments. For purposes of calculating the amount of data sent per feature, acknowledgments of data packets were included in the overall count.

The underlying communication mechanism provided by GroupKit was not analysed in terms of how it could be improved for wireless use. While changes to this component would likely yield benefits, it appears to be a very complicated task which would require substantial changes to the basic toolkit. This research is concerned mainly with application behaviour rather than the architecture that supports it. Thus, we focused on the higher-level group interface features which generated the traffic.

Telepointer. The telepointer was tested by moving the mouse continuously for several seconds to provide a measure of bytes sent per second of telepointer activity. On average, the telepointer position update procedure was called 511 times over a test period of 65 seconds, with 56996 bytes sent in total (112 bytes/call). The overall number of bytes sent is very large for all tests, because for every mouse motion event, Calliope sends an update of the new mouse coordinates to all conference participants.

Calculating an estimate of bytes/second generated by the telepointer is difficult, as the number of telepointer updates will vary depending on such factors as the speed at which the mouse is moved. However, using the average above as an example, the rate is 877 bytes/second.

This is an exceptionally high rate: if the telepointer were used over a 19.2 Kbps wide-area cellular network, then this one feature would consume approximately 36% of the available bandwidth. These results suggest that the current telepointer cannot be easily supported in a low-bandwidth environment.

Public Annotations. The public annotation feature was tested by composing a short note. The text insertion mechanism was shown to be a source of heavy overhead, as it used keystroke updating. This resulted in several packets being sent with just one character in them. With a packet overhead of 40 bytes (20 bytes for each of the TCP and IP headers), this is an unacceptably inefficient way to transmit data. The average data rate over the tests is 829 bytes/character, which represents an enormous amount of traffic for such short messages. An analysis of the traffic pattern revealed an implementation inefficiency which sent irrelevant data along with the message; this can be easily repaired. However, even when the redundant traffic is removed, the keystroke updates are still very expensive in terms of bandwidth used: each single character insertion (which is handled almost the same as in the main text window) sends 217 bytes.

External Annotations. The external annotation was tested by entering a 30-character URL. Unlike the public annotation, the characters are not sent one at a time. Thus, all the text is delivered at once, usually in one packet. Beyond the basic text for the URL, an external annotation carries an overhead of approximately 400 bytes, which is as efficient as possible because it consists of a single command sent over the channel.

Keystroke Updates. With immediate keystroke updating, every character is sent to all other users as soon as it is typed. This feature was tested by typing several characters in the main text window, so that each character was sent immediately.

The trace data reveals that for every insertion, the amount of data sent is approximately 216 bytes plus the number of characters inserted. With immediate keystroke updating, each character incurs this overhead. As typing is a very common activity in Calliope, this inefficiency is an important factor for wireless collaboration.

Private Text. The private text feature was tested with the same test as the keystroke update test above, except that the text was entered into the private text window, and thus was only sent when the user requested an insertion.

Not surprisingly, the results are dramatically different from the keystroke updates. There were only four packets sent: one to send initialization information, one with the whole text message, and one acknowledgment for each of these two packets. The block update data rate is only 2.3 bytes/character, which is 1.7% of the traffic that keystroke updates produce.

Group Scrollbar. The group scrollbar was tested by loading a large text file so that several screens of text appeared. One user scrolled from the beginning of the text to the end, while the second user followed this scrolling. Because the scrollbar updates are bound to mouse button presses and releases, more scrollbar updates can be sent than the user intends. If, for example, a mouse button were accidentally pressed and released twice on the same spot of the scrollbar, the screen view will not change, but an update will be sent. (This traffic analysis does not include the loading of the large file, only the group scrolling traffic.)

There was more traffic in this trace than anticipated, and the trace revealed an implementation inefficiency that sent redundant scrollbar updates. On average, 202 bytes were sent for each invocation of the scrollbar procedure: not a lot of traffic, but any excess calls would add up quickly.

Concurrency Control. Calliope supports four levels of lock granularity: selection, word, line, and paragraph. Each lock was tested by loading in a short text file, then selecting the lock type and grabbing the appropriate region. The amount of traffic sent is similar for most tests, except for the lock selection test. For locks that surround a pre-defined unit of text (word, line, paragraph), the amount of text sent is similar, as each lock request merely sends the appropriate set of lock start and end locations. The average amount of data sent, over all tests of word, line, and paragraph locks, is 375 bytes per lock request for every participant. Again, an implementation inefficiency related to the scrollbar was found, which if removed reduces the network traffic for a lock request to 227 bytes. This is fairly efficient (given the underlying communication mechanism), with similar amounts of data being sent over the channel regardless of region size.

The lock selection mechanism, however, generates more traffic than the other lock types, as it sends a new lock request for every change that the user makes in the selection range. Thus, to select a region for locking took an average over the two tests of 44 bytes *per character* in region selected. This will vary, depending on how the text is selected (how many changes to the lock region are made), but it is still a high per-character overhead.

Scratchpad. The scratchpad supports several types of unstructured activity, such as moving and editing text fragments and performing freehand drawing. These features make it difficult to create a standard test, as such testing actions as dragging text and drawing cannot be reproduced exactly in repeated tests. Instead, the tests demonstrate the amount of traffic generated by a single session of use. Each user manipulated the scratchpad in turn, following a standard script. Text (nine characters) was entered into the scratchpad, and this fragment was then dragged to the bottom of the scratchpad. Some lines were then drawn, and the text was erased and replaced with another 13 characters. On average, the total number of bytes sent was 15955. (A bytes/second rate calculated from this type of test is not useful, as it includes pauses and composition time in the total time.) Overall, the scratchpad sends a lot of traffic, mainly because of immediate updating of text and drawings.

Design strategies for adaptive groupware

The previous section demonstrated that groupware applications such as Calliope currently require substantial bandwidth, more than is commonly available in wireless networks. This section describes several general strategies that can be used to reduce this bandwidth usage. While this is not an exhaustive list, and use of these strategies is extremely application-dependent, it should provide a good starting point for a wide range of applications. In the following section, their specific application to Calliope will be shown.

Document Overview

Because a wireless channel has little available bandwidth, large data transfers take a long time to complete, which can increase user frustration. One such situation occurs when a user downloads a large file for editing. Often, the user does not intend to work on the whole text: she may want to update Chapter Three only, and thus a lot of unnecessary data is being sent.

What would be useful in this case is an overview of the document that allows the user to select only the relevant portions of text for the editing session. Markup languages such as SGML, HTML and LaTeX provide document structure within the text; such marks show how a document is partitioned, and are well suited to text filtering schemes. However, not all documents being edited will have been written in a markup language. Thus, there must be provision to deal with unstructured documents so that an overview can be created to filter irrelevant text.

Incremental Updates

Another source of heavy network traffic is frequent updating. Many groupware applications support a ‘‘What you see is what I see’’ (WYSIWIS) interface, which requires continual updates to ensure a consistent shared view [3]. Thus, every time a person changes a single character in the text, this change is immediately propagated. As each keystroke is encoded as a small amount of information, much of the traffic generated is packet overhead (mostly-empty packets). This sort of overhead may be far too expensive in a wireless environment.

Updates can be sent more efficiently in blocks of characters, so it might be better to send periodic updates of a block of text. It would also be useful for updates to be sent over the wireless channel when conditions are favourable (e.g., when the transfer will not interfere with a higher-priority networked application). However, this will lead to an inconsistent view of data for a short period. This could interfere considerably with a shared editing session, and so it should be invoked only when necessary.

Concurrency Control

Limited bandwidth and sudden disconnections must be considered when designing methods of concurrency control in mobile groupware. One very common method for concurrency control is the use of locks. If locks are used in a mobile environment, then a centralized lock manager should not be used, as it is not fault-tolerant. Instead, a replicated model could be used. Each user maintains a list of the currently-held locks. Whenever a user requires a lock, he can check his own copy of the list to see if the region is currently being held by any other user. If the region can be locked by this user, then he can send a serialized update to all other participants, informing them that this region is now under his control. Each user will then have a current list of all locks being held, a list which is updated each time a lock is released or granted.

This model may send little traffic (depending upon the specific implementation), as it only needs to send the single new lock update to other users. However, there are ways that it can send less traffic. One simple way to cut down on traffic is to reduce the number of lock updates sent. This could be done if the user was “greedy” in requesting locks. They could, for example, request a whole line if they thought they would need to change two words on that line, which allows one lock request instead of two.

An additional change that could reduce traffic is the creation of “roles” that would restrict the number of users involved in lock negotiation. If a user does not want to change the document, then she could be in “read-only’’ mode, which requires no locks. Thus, the lock request and subsequent release would not be sent to her machine.

Another issue that arises in concurrency control is disconnection. Locks must be released smoothly upon disconnection to allow other users reasonable access to the text. A timeout mechanism should be used to release locks after a period of inactivity.

Document Backup

Because disconnections can be frequent and sudden in a wireless system, a document backup system should be included in mobile groupware. Mobile users can be expected to disconnect, and their work could be lost if there is no stable storage with a recent copy of the document. Saving on demand is not sufficient in such an unstable system.

Due to the fact that people will want to be able to work even when disconnected from the session, local backups should be kept. A default “time between backups” could be set, with this time changing as network conditions change. For example, a high error rate suggests that a disconnection may happen soon, so a backup should be made to avoid losing data. This solution, however, requires that local copies must be merged into a consistent document. This is a difficult problem, but one that has been extensively studied: one example is a groupware package developed by Dourish [2] that provides mechanisms for document merges. There is also extensive applicable research on mobile computer file system consistency, such as the Bayou [11] and Coda [9] file systems. These systems address the concerns of data consistency for users who are periodically disconnected from a network.

Restricting Features

One final issue that must be considered is wireless support for group interface features. Several editors provide group awareness tools to give individual users a better sense of the working group. While these tools are valuable, they may prove too expensive under poor network conditions. For example, a feature may generate frequent updates, such as an overview that displays each user’s position in a document. This feature could be turned off or scaled back temporarily if there was insufficient bandwidth to support it.

Any editor that has a bandwidth-intensive feature might want to provide a less resource-consumptive version for use on mobile hosts. This could involve fewer updates or a less complete interface, depending on the feature. In either case, it would likely be better to allow limited use of a feature rather than having to disable it completely to allow the editor to work.

Application of design strategies

The strategies described above were evaluated with respect to Calliope. What follows are suggestions for how the present features of Calliope could be adapted for more efficient use of wireless channels. The suggested changes were evaluated by extrapolating their effects to the observed trace data; actual implementation was outside the scope of this research.

Update Frequency

One change that was evaluated was to alter the frequency of updates to the shared document view. Calliope uses keystroke-by-keystroke updating in order to maintain a WYSIWIS interface. Block updates should be implemented instead of character updates. These blocks can be of varying lengths, depending on network conditions or user preferences. Sending a whole word, sentence or an even larger block of text will cut down on the number of small packets being sent over the wireless channel. Alternately, changes can be sent periodically, for example, every three seconds. Yet another way to implement block updates would be to use a locked region as a block of text, the block updated only when the lock is released. With any of these approaches, keystroke updating could resume when bandwidth levels increased.

Concurrency Control

A second change that was evaluated was to Calliope's method of concurrency control. Calliope uses a replicated locking mechanism, as described above, which is fairly efficient as implemented. The selection lock is the only lock which can be too expensive for a wireless environment. As a quick fix, this type of lock could be disabled when available bandwidth was low; the user would be forced to lock only a word, line, or paragraph. Alternatively, a number of lock requests could be bundled together and sent after a short interval. (For information on how these changes might affect the interface, the research by Greenberg and Marwood [5] provides much useful data.)

Group Interface Features

Many changes were examined that would make Calliope's group interface more adaptive. Calliope has four central group interface features that could be disabled or scaled back: the telepointer, group scrollbar, unstructured text space, and annotations.

Calliope already provides for the telepointer to be enabled and disabled by the user, but there should be some middle ground that allows this tool to be used under poor conditions. Limiting the number of telepointer updates that are sent would reduce network traffic, allowing the telepointer to be used when conditions are slightly less than ideal. There may be a lower sampling rate that allows reasonable position updates, such as sending an update for every third or fourth mouse motion event, which would allow the telepointer to be supported without overwhelming the wireless link. In the very worst circumstances, where the telepointer traffic interferes with more important traffic (such as receiving text), then it could be temporarily disabled, then enabled to a limited degree when conditions improved.

A more complicated solution involves replacing the telepointer with a different type of semantic cursor, one that is less bandwidth-expensive. The creators of GroupKit have proposed some alternatives to the telepointer, such as sending a "point request" by clicking a mouse button over selected text [6]. Similar alternatives, such as selecting a text region of interest, would allow users to maintain the ability to gesture even under adverse network conditions.

Another set of tools which could be adapted are the external and public annotations. Both annotations are written into a small window whose contents are inserted peripherally into the text: they can be made visible by clicking on an icon. External annotations contain a URL which can be followed by spawning a web browser, while public annotations consist of notes made by one participant intended to be read by others (such as suggestions for word changes). The annotations are very simple features, and as such, are difficult to modify for wireless use. The external annotation can either be used or not, depending on channel quality, but it cannot be restricted any further. The public annotation is similar, but it uses keystroke updating, which could be replaced by block updating to cut down on the number of text updates sent over the wireless channel. This change is the most efficient possible without entirely losing the ability to create public annotations.

As with the annotations, the group scrollbar cannot be easily converted into a less resource-consumptive version. A reduction in the number of scrolling updates will reduce network traffic, but this can only be done in limited circumstances. For example, if a user wanted to scroll through a document until they found a certain passage, and felt that the intermediate scrolling did not need to be shared, then they could send an update only when they had arrived at the screen they wished to work on. When one person scrolls through a long text file, it may be irrelevant for other parties to be following along. Once the right section has been found, the other participants can be sent the proper coordinates so that

	Suggested Changes Under Given Condition		
	Degraded	*Slow*	*Extremely Poor*
Public Annotation	No change	Use block updates	Use sparingly
External Annotation	No change	No change	Use sparingly
Telepointer	Decrease number of updates	Disable	Disable
Group Scrollbar	No change	No change	Send only final scroll or bundle requests
Locks	No change	No change	Request large locks or bundle requests
Scratchpad	Decrease number of updates	Use block updates	Disable
Keystroke Updates	Send few characters at once	Send more characters at once	Send entire block at once

Table 1: Summary of Calliope features and suggested changes

they can share the same view, or scrollbar coordinates could be sent periodically over short intervals. This would cut down on updates that serve little purpose. If it became necessary, this feature could be temporarily disabled under adverse network conditions.

The final tool to be considered is the scratchpad, which allows for cutting and pasting text from the main document into the scratchpad, for writing and moving text, and for freehand drawing with the mouse. Each of these features can be restricted in some way, and their similarities with other features described above suggest that some of the same solutions proposed can be used here. The scratchpad, like the main text window, uses keystroke updates, which could be replaced with block updates. The drawing and dragging features use mouse motion events to generate updates, and thus suffer from the same problems as the telepointer. Fewer updates, as few as can be allowed without degrading the interface too severely, would sharply reduce traffic. These changes may allow the scratchpad to be used when bandwidth drops, although for very poor network conditions, it is likely that the feature could not be used at all.

Summary

Table 1 gives a summary of the features that are already implemented in Calliope, along with suggestions for how they could be made adaptive in order to perform better under various levels of connection quality. These general levels (e.g., “slow”) are used because the exact conditions for triggering adaptation were not formally defined or evaluated. Because the usability of the group interface can suffer as features are removed, it is a difficult process to discover the exact point at which a change in features provides the greatest gain for the entire application. Table 1 provides a general guide to show relative gains among feature changes, which demonstrates that a feature does not have to be removed in order for an application to reduce its consumption of network resources. Designers can select the adaptive response that best serves the needs of their application.

Performance improvement of adaptive Calliope

In order to evaluate the adaptive version of Calliope, a theoretical scenario was developed in which both the original and the proposed adaptive editors were used to perform the same actions. This scenario was used to compare traffic generated by both models, as well as to provide a sense for what the user might experience while using an adaptive editor. Because the adaptive editor was a proposal (rather than real software), its communications costs were based on extrapolations from the actual test data gathered on Calliope’s channel usage.

Test Scenario

Shown below is a scenario in which Alice and Bob are collaborating on a short memo. The wireless LAN that they share degrades from good quality to fairly poor quality (“slow”) after the first editing action. A number of actions, detailed in Table 2, are conducted on the text in the memo below, which was composed prior to the start of the session.

```
Internal Memorandum

From           : Head of Supplies
To             : All Departments
Subject        : Change in Order Forms
There has been a recent change in the Office
Supply Ordering Form, which is now Form I-6785
rather than Form I-7856. All those who wish to
order supplies must now use Form I-6785,
including any person wishing to order Office
Supply Ordering Forms (Form I-6785).

Note that all copies of obsolete Form I-7856
will be collected at the end of the day.
Following departmental regulations, any person
found in possesion of Form I-7865 after this
date shall be destroyed immediately along with
the requisitioned forms.
```

The actions taken on this text are as follows:

User Action	Subactions
1. A: changes ‘immediately’ to ‘summarily’	9 chars. deleted, 7 inserted
2. B: non-adaptive: requests *line* lock to change ‘who wish to order’ to ‘wishing to order’. Adaptive: requests *paragraph* lock to change same text both: lock is granted and text is changed	1 lock request, 4 chars. deleted, 3 inserted
3. A:changes ‘posession’ to ‘possession’	1 char. inserted
4. B: non-adaptive: requests line lock to change ‘any person’ to ‘all persons’; lock is granted. Adaptive: no new lock required both: text is changed	non-adaptive only: 1 lock request. both: 2 chars deleted, 3 inserted
5. A:uses public annotation to send “Replace ‘departmental regulations’ with ‘official procedures’?” as suggestion to Bob	62 chars. inserted in public annotation
6. B” requests line lock to change ‘departmental regulation’ to ‘official procedures’; lock is granted and text is changed.	1 lock request, 24 chars. deleted, 19 inserted
7. A: adds ‘office’ before ‘supplies’ in main text	6 chars. inserted

Table 2: User actions in test scenario

Analysis of improvement

For the actions shown in Table 2, we can construct an estimate of the traffic sent. Three models are compared below: the original implementation, an improved (but non-adaptive) version which removes the redundant data transfers found in the original, and the improved version with adaptability added. As detailed below, the improvements to Calliope are substantial. By removing redundant traffic, bandwidth consumption drops by 55%; by adding adaptability, the bandwidth used drops by over 90% of that used by the original application.

For both non-adaptive versions, every character update carries an approximate overhead of 216 bytes plus the number of characters. The traffic analysis for the more efficient version assumes that the cost of an annotation text update is the same as an update in the main text window. The annotation is more costly in the original implementation, with an average cost of 829 bytes per character. Finally, the original Calliope lock requests require an average of 375 bytes, while the improved version requires only 227 bytes.

The resulting communication costs are shown in Table 3. The improvements make an enormous difference to the amount of traffic generated: the more efficient (but still non-adaptive) version generates only 45% of the bytes used by the original Calliope. As the data being sent in the original version is redundant, there are no detrimental effects caused by removing it from the communication channel. This scenario demonstrates that such inefficiencies can have serious effects on the usability of an application when it is to be used in a resource-poor environment.

The situation is much improved when adaptive Calliope is used. After the first action, when the channel quality drops to slow, the successive actions are modified to suit these new conditions. The keystroke updates are no longer used: instead, an update is sent for every tenth change. (A timeout could be used for changes smaller than ten characters.) The annotation now uses a block update mechanism for the entire text, so Bob will not see the annotation text until Alice finishes typing it. Since the annotation is not so important that he needs to follow along with its development, this is not a significant reduction in the feature’s usefulness. Finally, Calliope will recommend that the user be optimistic and select large locks if they will be making multiple changes to a region. Thus, Bob requests a paragraph lock at action 2, as he intends to change other parts of this region (action 4).

These changes lead to the results in Table 3. The total traffic sent by adaptive Calliope is approximately 21% of the traffic sent by the non-adaptive version (column 2) and, in this case, there is not a significant reduction in the usability of the editor. For the price of making users wait a short time for updates (and possibly interfering with concurrent text changes in a region) the number of bytes sent is dramatically reduced. This result suggests that the adaptive version of Calliope makes a reasonable trade-off between interface features and

Traffic (in bytes) generated by original and improved models of Calliope		
Original	*Improved and Non-Adaptive (Percentage of Original)*	*Adaptive (Percentage of Original)*
3472	3472 (100%)	3472 (100%)
1894	1746 (92.2%)	450 (23.8%)
217	217 (100%)	217 (100%)
1460	1312 (89.9%)	437 (29.9%)
51398	13454 (26.2%)	278 (0.5%)
9706	9558 (98.5%)	1350 (13.9%)
1302	1302 (100%)	222 (17.1%)
total: 69449	**total: 31061 (44.7%)**	**total: 6426 (9.3%)**

Table 3: Performance gains for two improved models of Calliope

bandwidth consumption, and is a suitable model for editors being used over a wireless channel.

Related work

Recent work in mobile computing has examined the necessity for adaptive applications. Carnegie Mellon University's Odyssey project is designed to allow applications to negotiate for network resources [10]. The Odyssey project gives no guidelines for application adaptation, so this research on adaptive groupware would be of use to those who wish to use Odyssey but are unsure how to scale back a group editor when they are granted few resources.

Terri Watson designed an adaptive web browser, W*, which adapts to variations in network connectivity and provides the user with information about potentially slow hyperlinks [12]. However, there are no changes to the interface to make better use of limited resources: W* might benefit from a more adaptive interface, and some of the changes proposed in this paper might be applied or modified for use on a browser.

Extensive research has been conducted into the problem of maintaining consistent documents in file systems where users are frequently disconnected. The Coda system [9], developed at Carnegie Mellon University, was designed to support consistency in the presence of disconnected operation through the use of cached storage and replicated servers. Xerox PARC's Bayou system [11] provides similar support, using domain-specific knowledge to detect and resolve conflicts on shared data.

At the University of Tromsø, investigators are studying a diary application which downloads varying amounts of information based on the quality of the current connection [4]. As with Watson's browser, changes to the interface are minimal. Thus, there are many issues discussed in this paper that could be used to make the diary application interface consume fewer resources when necessary.

Some work has been done in the collaborative software in a mobile environment. Koch's IRIS group editor [7] allows users to request or send updates on demand, which is particularly useful for transfers over variable wireless channels. However, Koch's research focuses mainly on an interface that adapts to different user requirements, rather than connection quality, and provides few recommendations for ways of restricting the interface. If Koch's model were integrated with this research on specific interface changes, then a very adaptive, robust mobile editor model would result.

The most comprehensive research on adaptive, mobile groupware has been conducted at Lancaster University [1]. Lancaster have developed a prototype to allow field engineers to use a geographical information system to view such material as network diagrams. The application is informed of changes in connection quality, and adapts its behaviour to make better use of the resources available. Their research, however, deals with a specific application, and does not provide design principles that can be extended for use with other types of groupware. The basic design issues described in this paper can be used to extend their work so that other groupware applications can work effectively with their adaptive architecture.

Conclusions and future work

This research demonstrated that while current groupware applications are unsuitable for use in wireless networks, these applications can be modified to work more efficiently over wireless channels. In particular, our analysis of the bandwidth requirements of Calliope showed how its bandwidth usage could be drastically reduced without restricting the user interface unnecessarily.

We described a number of general strategies for building adaptive groupware which should be applicable to a variety of applications. These strategies include providing document overviews and backups, sending incremental updates, streamlining concurrency control, and restricting user interface features. These techniques allow an application to make effective use of a variable and often poor wireless channel. Implementing even a few simple changes can result in considerable reductions in bandwidth consumption; with a small amount of programming effort, existing applications can become usable by mobile collaborators.

This work provides a starting point for research in building groupware that adapts to wireless connections. There are many areas where further research is necessary.

This research considered a single groupware application. Other applications need to be investigated, both to validate and extend the set of design strategies presented here. Further study could also lead to the inclusion of adaptive components in general purpose groupware toolkits.

In order for adaptation to be supported, an application must have a means of obtaining information about network conditions. It would be helpful if a generic mechanism were available in a toolkit, one that would supply a generic and configurable source of information that several types of applications could use to support adaptive behaviour.

Finally, usability studies need to be conducted on these adaptive systems to reveal the effects that these changes will have on the usability of groupware applications. For example, one major issue to be resolved is how to handle adaptation. It is unclear whether adaptation should occur automatically or whether a user should approve these changes.

As wireless networks become increasingly widespread, the demand for efficient, responsive groupware for these networks will also increase. To meet users' needs, groupware systems must be able to adapt effectively to the challenges of a wireless environment.

Acknowledgments

Tara Whalen would like to thank Mike Nidd for technical support of this research and Mark Roseman for many helpful comments on this paper. The authors also thank the National Sciences and Engineering Research Council (NSERC) and Motorola Canada for financial support.

References

1. Davies, N., Blair, G.S., Cheverst, K. and Friday, A. A collaborative multimedia application for a mobile environment. Technical Report MPG-94-14, Lancaster University, 1996.
2. Dourish, P. The parting of the ways: divergence, data management and collaborative work. In *Proceedings of the Fourth European Conference on Computer-Supported Cooperative Work ECSCW '95*, Stockholm, Sweden, September 1995.
3. Ellis, C.A., Gibbs, S.J. and Rein, G.L. Groupware: Some issues and experiences. *Communications of the ACM*, 34(1):38-58.
4. Fallmyr, T. Adaptable mobile systems. In *Proceedings of the Fifth IEEE Workshop on Future Trends in Distributed Computing* (Cheju Island, Korea, August 1995), 363-368.
5. Greenberg, S. and Marwood, D. Real time groupware as a distributed system: Concurrency control and its effect on the interface. In *Proceedings of the ACM 1994 Conference on Computer Supported Cooperative Work* (Chapel Hill, North Carolina, October 1994).
6. Greenberg, S., Gutwin, C. and Roseman, M. Semantic telepointers for groupware. In *Proceedings of the OzCHI '96 Sixth Australian Conference on Computer-Human Interaction* (Hamilton, New Zealand, November 1996).
7. Koch, M. Design issues and model for a distributed multi-user editor. *Computer Supported Cooperative Work — An International Journal*, 3(3—4):359-378.
8. Mitchell, A. Communication and shared understanding in collaborative writing. Master's thesis, Department of Computer Science, University of Toronto, 1996.
9. Satyanarayanan, M. Coda: A highly available file system for a distributed work environment. In *Proceedings of the Second IEEE Workshop on Workstation Operating Systems* (Pacific Grove, California, September 1989).
10. Satyanarayanan, M. Mobile information access. *IEEE Personal Communications*, 3(1):26—33.
11. Terry, D.B., Theimer, M.M., Petersen, K., Demers, A.J., Spreitzer, M.J. and Hauser, C.H. Managing update conflicts in Bayou, a weakly-connected replicated storage system. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles (SOSP-15)* (Copper Mountain Resort, Colorado, December 1995).
12. Watson, T. Application design for wireless computing. In *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications*, (Santa Cruz, CA, December 1994), 91—94.